Automatic Analysis of Computation in BioChemical Reactions

Attila Egri-Nagy, Chrystopher L. Nehaniv, John L. Rhodes[+], Maria J. Schilstra

The Royal Society/Wolfson Foundation BioComputation Laboratory & Algorithms Research Group
Centre for Computer Science and Informatics Research
University of Hertfordshire
College Lane, Hatfield, Hertfordshire AL10 9AB, United Kingdom
{A.Egri-Nagy, C.L.Nehaniv, m.j.1.schilstra}@herts.ac.uk

[+]*Department of Mathematics*
University of California, Berkeley
Berkeley CA 94720 USA
rhodes@math.berkeley.edu

Abstract

We propose a modeling and analysis method for biochemical reactions based on finite state automata. This is a completely different approach compared to traditional modeling of reactions by differential equations. Our method aims to explore the algebraic structure behind chemical reactions using automatically generated coordinate systems.

In this paper we briefly summarize the underlying mathematical theory (the algebraic hierarchical decomposition theory of finite state automata) and describe how such automata can be derived from the description of chemical reaction networks. We also outline techniques for the flexible manipulation of existing models. As a real-world example we use the Krebs citric acid cycle.

1. Introduction

Here we propose a modeling and analysis method for biochemical reactions based on the algebraic hierarchical decomposition theory of finite state automata (also known as Krohn-Rhodes Theory [1]). This method can be used in conjunction with the classical modeling methods based on differential equations, as they each work on a different level of abstraction. The automata based method analyses the discrete computational structure of the reaction network, while differential equations model the actual physical processes. Clearly, we have to resolve several issues regarding the possibility of a sensible finite state description of a reaction network.

The ultimate goal is to provide a classification of biochemical reactions along the lines of the classification of well-studied algebraic structures (finite simple groups) and to describe such reactions using predictive coordinate systems that are automatically constructed and based on algebraic structures (transformation semigroups) canonically associated to reaction graphs. The finite simple groups are now generally accepted as having been fully classified (cf. [2]) as a result of a concerted effort, spanning well over a hundred years, of numerous researchers in group theory. Therefore the reactions could be potentially studied in a way that exploits this extensive mathematical groundwork.

Most of the ideas described here were proposed long time ago in a (still unpublished) book by John L. Rhodes, usually referred as the “Wild Book” [3], developing ideas presented earlier [4] on applying automata theory to biochemical reactions. But recently we have created the first computational tools [5,6] realizing these methods (in particular the holonomy method of obtaining decompositions used here – see, e.g. [7, Chapter 3]), and therefore the exploitation of these ideas can begin now. We have also begun to apply the ideas of automatically generated coordinate systems in other fields: for genetic regulatory networks [8] and in artificial intelligence [9].

2. Reactions as Automata

We have tools for understanding any phenomenon in terms of coordinate systems (see Sect. 3), if it is amenable to a finite state automata description. However, biochemical reaction networks consist of macroscopically continuous processes, their state descriptions involve concentration values and thus they are quite far from a discrete model (although individual molecules are in fact discrete entities). Therefore their modeling is usually done by differential equations. Differential equations are defined in a recursive form, one has to actually calculate their time evolution in order to gain some knowledge about the system (and even this provides no, or only limited, insight as to important factors, organizing principles, etc.).

In order to generate a finite state automata description, we can try discretizing the continuous state space and time. But this approach has several problems. One is the combinatorial explosion which may arise during discretization if the the state space has many dimensions. Moreover, it may be possible that we thus abstract away important properties (like subtle changes of concentrations triggering other events) of the reaction network. Therefore we propose a completely different approach.

2.1. *Perturbation-based Modeling*

First we need to explain some biochemical terminology. *Enzymes* are biological catalysts. Catalysts modify the speed of a chemical reaction without being used up or appearing as one of the reaction products. The reactants (i.e. the molecules or molecular assemblies that are consumed) in a particular enzyme-catalysed reaction are called the enzyme's *substrates/metabolites.* These are molecules, usually of low molecular weight with respect to enzymes, that take part as reactants and products in metabolism, the complete set of chemical reactions that occur in living cells. Individual metabolites usually take part in a limited number of different reactions. In the examples presented here, in the Krebs cycle, pyruvic, citric, isocitric, oxaloacetic, α-glutaric, succunic, fumaric, and malic acid are the metabolites. A *coenzyme* is a special type of substrate, one that fulfils the same function, that of the acceptor or donor of a specific chemical group, in many different reactions. Coenzyme acceptor/donor pairs that take part in the Krebs cycle are NAD^+/NADH, $NADP^+$/NADPH, CoA/acetyl-CoA (where CoA is an abbreviation for CoenzymeA), and FAD/FADH. We also make use of the intuitive notion of a *soup*, which is a solution containing all the required substrates, inorganic ions and enzymes required for the reactions.

The idea is simple:

Idea 1 *Steady states of the system are the states of the automaton, and perturbations to these steady states are the inputs of the automaton (rather than time intervals).*

The definition of the steady state can be a matter of dispute but generally it corresponds to some macroscopically observable stable circumstance (e.g. fixed concentration levels of substrates). The input can be a sudden change of the concentration level of any substrate, enzyme, inorganic ion. However, here we choose a slightly more restricted approach here. We choose coenzymes as inputs; all the substrates, inorganic ions, inactive enzymes are in the soup. So we choose inputs as the "last missing tiny bits" that are required for the reaction.

Thus in our model we will not attempt to describe such phenomena as membrane transport or diffusion, although such physical aspects are important, perhaps crucial, to the maintenance of many of the metabolic states. Another aspect of our model is that it is not kinetic, i.e. it will not be concerned with rates. The model will be built on the basis of biochemical reactions that are known to occur, but whose rates need not be known, especially when they occur as part of a complex system of reactions. Simplifications are usually necessary, sometimes even desirable, in any analysis, however distortions (and falsifications) are to be avoided as much as possible. Accordingly, the model has been conceived as being embeddable in a real — *in vitro* — situation.

2.2. *Example: The Krebs Cycle*

Under normal circumstances the major portion of the cell's energy requirements are met by the breakdown of sugars.

The principal source of energy for the cell is glucose and the complete burning of 1 mole of glucose via the Krebs cycle is given by

$$C_6H_{12}O_6 + 6O_2 \rightarrow 6CO_2 + 6H_2O + 673 \text{ kcal}.$$

In the cell, glucose is converted to CO_2 and water by a process involving nearly 30 different steps and in each step a small amount of energy is realized. This reaction network was a big evolutionary leap, since anaerobic glycolysis is much less efficient at extracting energy from sugar [10].

The Krebs cycle is well understood in its fine details, therefore it is an excellent example for demonstrating and validating our methods. Any standard textbook in biochemistry contains the details (see e.g. [11]).

Figure 1 shows the reaction graph of the Krebs cycle in which substrates appear as states and arrows are labelled by coenzymes required for the transition. In cases where the transition from one substrate to another runs to completion given the constitution of the soup, the substrates in questions are shown as in the same state of the model (e.g. Citric and Isocitric Acid). We denote this model of the Krebs cycle as K_1. Regarding the states as states of a finite automaton and the transitions as the action of inputs, we will be in a position to apply algebraic automata theory to obtain a decomposition and coordinate system for this reaction system, once we introduce some basic concepts in the next section.

3. Coordinate Systems for Finite State Automata

3.1. *Finite State Automata*

By a finite state *automaton*, we mean a triple $\mathcal{A} = (A, X, \delta)$ where A is the (finite nonempty) *state set*, X is the *input alphabet* and $\delta : A \times X \rightarrow A$ is the *transition function.* We do not explicitly consider any output of the automaton, as any such function can be recovered from the state and the input symbol. We tacitly use the state as the output.

For example, in model K_1 of the Krebs cycle (see Figure 1), $\delta(a_5, x_4) = a_6$ since the reaction graph shows

$$a_5 = \text{Succinyl CoA} \xrightarrow{x_4=\text{GDP}} \text{Succinic acid} = a_6,$$

whereas

$$\delta(a_1, x_4) = a_1$$

since, according to the model there is no arrow labelled GDP (x_4) leaving state a_1, so if GDP does come in contact with pyruvic acid (a_1), it leaves this substrate unchanged.

As the state transition function δ is fully and uniquely defined for all possible state and input symbol combinations, we have a complete knowledge of the automaton. However, this knowledge does not imply immediate understanding of the automaton, as it is given in a *recursive form* (just as in the case of differential equations), i.e. in order to gain insight about the computations carried out by the automaton, we need to start from an initial state, apply an input symbol, arrive in another state, then apply another transformation, and so on. Formally, we can naturally extend the transition function to input “words”, i.e. sequences of input symbols: for the empty word $\delta(a, \lambda) = a$, and for arbitrary words $u, v \in X^*$ (finite input sequences from the input alphabet X), we have $\delta(a, uv) = \delta(\delta(a, u), v)$. This justifies the practice that, for notational simplicity, we sometimes write $x \cdot u$ instead of $\delta(a, u)$ for any $a \in A$ and $u \in X^*$. For example, in Figure 1, $\delta(a_1, x_6x_1x_2) = a_1 \cdot x_6x_1x_2 = ((a_1 \cdot x_6) \cdot x_1) \cdot x_2 = a_3$ as the reader can immediately verify.

Thus we can follow a trajectory through the state space by applying input sequences starting at any given initial state, but in order to promote understanding we need to introduce coordinates into the state space.

3.1.1. *Transformation Semigroups*

Finite state automata can be looked at in an algebraic way. Each input symbol gives an operator on states, denoted simply by x where $x \in X$. For instance x_2, the operator associated to CoASH in the model of the Krebs cycle K_1 changes states as follows:

$$\begin{array}{ll} a_1 \mapsto a_1 & a_5 \mapsto a_5 \\ a_2 \mapsto a_3 & a_6 \mapsto a_6 \\ a_3 \mapsto a_3 & a_7 \mapsto a_7 \\ a_4 \mapsto a_4 & a_8 \mapsto a_3 \end{array}$$

since CoASH acts nontrivially only on substrates Acetyl CoA (a_2) and Oxaloacetic Acid (a_8) in the Krebs cycle.

Several input symbols in a sequence also give a well-defined operator, namely the composition of their transformations. This way the input symbols generate a set of transformations, which is a *semigroup* S (i.e. a set equipped with an associative multiplication). If we keep the original state set A of the

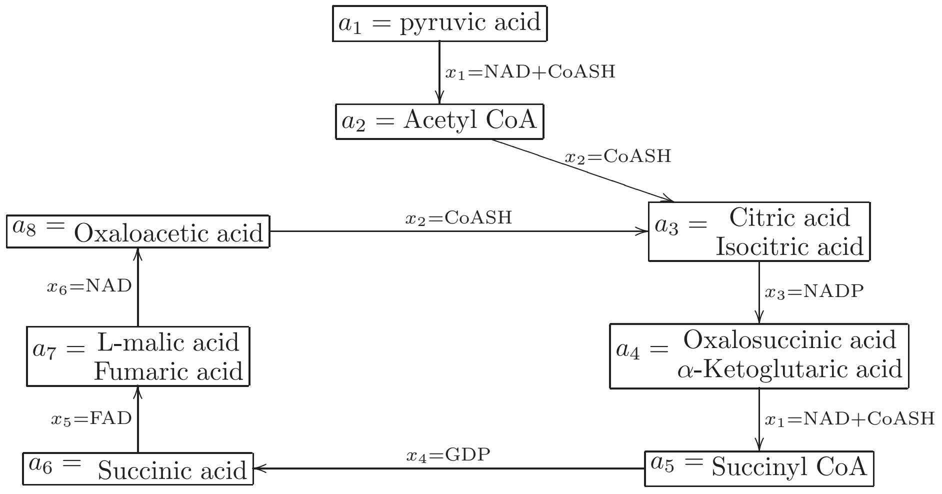

Fig. 1. Reaction graph and finite state automaton model K_1 of the Krebs cycle. Substrates corresponds to states of the automaton and if acted on by the relevant coenzyme are transformed, e.g. pyruvic acid (a_1) interacting with NAD and CoASH (x_1) is transformed into Acetyl CoA (a_2). This is denoted $\delta(a_1, x_1) = a_2$. Note that in case the coenzymes have no effect on a substrate then no arrow is shown, e.g. $\delta(a_8, x_4) = a_8$ since Oxaloacetic acid (a_8) is not transformed by interaction with GDP (x_4). The presentation is "event-based" or "perturbation-based" and time does not appear explicity in the model. Reactions shown run to completion, and thus all transitions are idempotent (i.e. $\delta(a, x) = \delta(\delta(a, x), x)$ always holds in a reaction graph).

automaton with the semigroup, then we get a *transformation semigroup* denoted by (A, S). The transformation semigroups should be considered here as just a different view for finite automata. Moreover, if the transformations are all permutations (one-to-one mappings) we talk about a *permutation group.*

3.2. *Algebraic Hierarchical Decomposition*

For explaining Krohn-Rhodes Theory, the best way is to present it by using a metaphor. Basically we do the same thing that the decomposition into prime factors does for the integer numbers, but instead of for integers we do it for more complicated structures, namely finite state automata considered as transformation semigroups (see Fig. 2).

The basic building blocks are (1) the simple [1] permutation groups (for the reversible computation) and (2) a single additional building block for the irreversible computation, the so-called *flip-flop automaton*, which is essentially a one-bit resetable memory that can be set and read. [2] The simple groups are called the "*primes*" of this theory, as they play a role analogous to the prime numbers in the multiplicative decomposition of integer numbers (see Figure 2 and the discussion of this analogy below).

The way of putting together the components, the so-called *cascaded* or *wreath* product, is hierarchical and no feedback is allowed from deeper levels to upper levels (see Fig. 3). The usefulness of this special type of composition is due to the following special properties of hierarchy that render the composed structure manipulable and comprehensible:

– Generalization and specialization are natural operations realized by taking subsets of levels in either direction up or down the hierarchy.
– Information flow between levels is restricted, avoiding problems associated to understanding feedback.

Note that any number of parallel, non-interacting components are allowed on any hierarchical level.

The hierarchical composition is a proper balance between the two conflicting requirements: having a nice, comprehensible structure and possessing the expressive power to construct any arbitrary automaton (see Fig. 4). The parallel composition (direct product) has a very simple structure, all of its components are completely independent, but this also means that it is not possible to surpass the complexity of the building blocks. On the other hand, if we

[1] This has a well-defined meaning in group theory: a group is *simple* if it has only trivial homomorphic images, i.e. any structure preserving map to another group is either one-to-one or collapses all elements to a single point. See, e.g. [12].

[2] An important but subtle point here is that although the flip-flop can be reset, this does *not* make it reversible. Indeed, it is not possible to reverse a resetting operation since this erases the previous state, and hence is not a permutation of the flip-flop's state set.

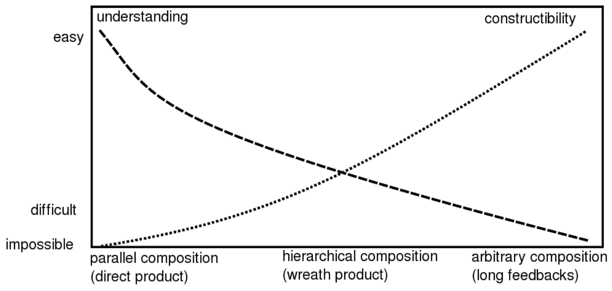

Fig. 4. Schematic diagram of the position of the wreath product relative to alternative decomposition methods. The direct product is the easiest to understand (just understanding the parallel components separately), but not all automaton can be constructed that way. When arbitrary wiring of the building blocks is allowed, the structure becomes impossible to understand. The wreath product seems to be the most satisfactory compromise between the two opposing criteria.

allow arbitrary wiring of the components (feedback loops with different lengths) then any system is realizable (even using only flip-flops as building blocks – see, e.g. [7]), but such a construction generally provides no insight into structure or dynamics. In contrast, with a cascaded decomposition, feedbacks in the original automaton being decomposed can give rise to structural permutation groups – possibly at different levels in the hierarchical decomposition – reflecting local symmetries in the system.

3.3. *Examples of Coordinate Systems*

The natural example of a hierarchical coordinate system is our decimal positional number notation system: different coordinate positions correspond to different power-of-ten magnitudes.

Another simple, but non-trivial example to describe *hierarchical dependence* is a bidirectional counter. Imagine a device which keeps track of how many times you press a button, where you also have two other buttons to set the operating mode. For instance to count and double-check the number of passengers on an airplane while walking along the aisle, you start from zero in adding mode, count, and then as a check whether the resulting number is the correct value, you switch to subtracting mode and count again, but this time downwards, until you reach zero again. The operation of this device can be represented with the following simple coordinate system on its states:

$$(n, \mathrm{mode}),$$

where n is the current tally and the possible modes $+$ and $-$ correspond to adding and subtracting. The mode coordinate is the top level of the hierarchy. The buttons provide three operations: counting c, switching to adding mode m_+, and switching to subtracting mode m_-. For instance, the result of each elementary operation is exemplified as follows:

$$(9,+)\cdot c = (10,+)$$
$$(9,+)\cdot m_- = (9,-)$$
$$(9,+)\cdot m_+ = (9,+)$$
$$(9,-)\cdot c = (8,-)$$

Hierarchical dependence here is clear: the counting operation does different things (adding or subtracting 1) to the coordinate giving the current tally, depending on the top level coordinate (the right coordinate giving the current mode); but this dependence is only one way: the state of the tally count (left coordinate) never influences the effect of the basic transformations on the mode coordinate.

4. Prime Components and Automatic Decompositions of the Krebs Cycle Models

We used our software tool `jSgpDec` [5] to decompose the transformation semigroups of model K_1 of the Krebs cycle and a simpified version K_2 (see Section 5) and to obtain their group and prime components.

4.1. *Decomposition of the Simplified Model K_2*

The prime group components of K_2 are the cyclic groups C_2 and C_3 of order 2 and 3, respectively. These correspond to the graphical cycle of the reaction network graph of the model. We have $PRIMES(K_2) = \{C_2, C_3\}$. The computational results confirm the correctness of the manual calculation of the "Wild Book" [3]. The decomposition is shown in Figure 6. The non-trivial permutation groups occur at the two lowest levels of the hierarchy.

4.2. *Decomposition of the Full Model K_1*

Moreover, we also decomposed the other more complex model of the Krebs cycle (the model K_1 shown in Figure 1) using the holonomy method. Carrying out this latter computation is far beyond the possibility of manual, brute-force calculation), and here we found the the same type of groups C_2 and C_3, but also the cyclic groups C_4 and C_5 with larger

	Integers	**Finite Automata**
Building Blocks	Units (± 1), Primes	Flip-flop Automaton, Permutation Automata
Composition	Multiplication	Wreath Product
Precision	Equality	Division, Emulation
Uniqueness	Unique (up to units)	Different Decompositions

Fig. 2. The parallel between the factorization of integer numbers and finite state automata. The very same idea is applied to computational structures in algebraic automata theory. Here we have two types of irreducible building blocks (also called *primes* (simple groups) and *units* (divisors of the flip-flop)), but as automata are more complicated structures, we have *emulation* of the decomposed structure by possibly larger, decompositional structure (also called *division* of the decomposition by the original automaton) rather than equality, and the factorization need not be unique at all.

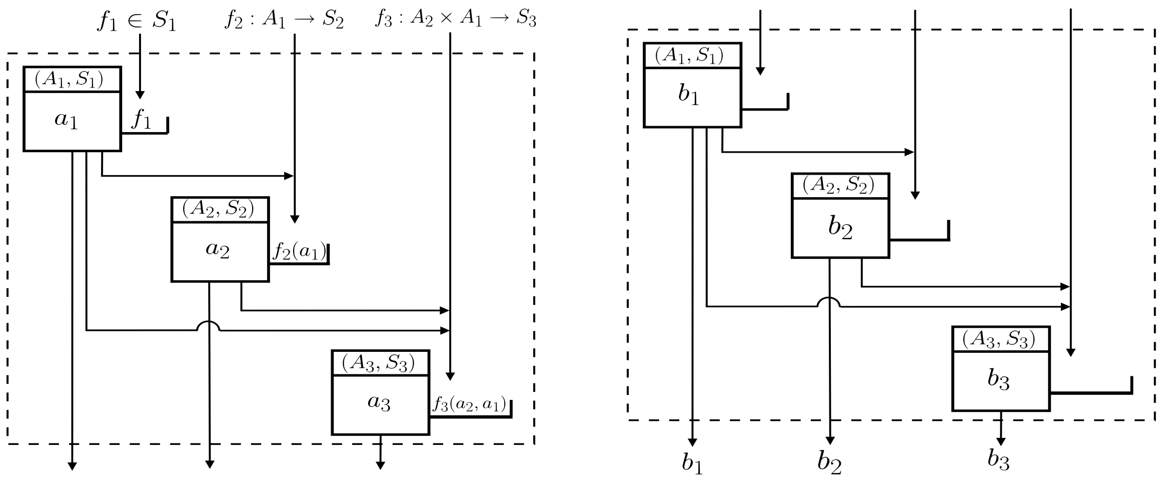

Fig. 3. Example of a 3-level coordinate system composed using the cascaded/wreath product of component transformation semigroups (A_n, S_n), $n \in \{1, 2, 3\}$. The resulting composed automaton is enclosed in dashed lines; both its input and output are 3-tuples. Left: For a state transition in the wreath product $(A_3, S_3) \wr (A_2, S_2) \wr (A_1, S_1)$, the input transformation (f_3, f_2, f_1) is applied to state (a_3, a_2, a_1) yielding $(b_3, b_2, b_1) = (a_3 \cdot f_3(a_2, a_1), a_2 \cdot f_2(a_1), a_1 \cdot f_1)$. The "trays" visualize how at each level i the components of the input (dependency functions f_i) are evaluated according to hierarchical dependence on the states at higher levels. The resulting transformations $f_i(a_{i-1}, \ldots, a_1) \in S_i$ are then applied to transform the state component within level i. Note that the applications of these functions happen simultaneously; their arguments are the previous states of other components, therefore there is no need to wait for the other components to calculate their new states. Right: The new state (b_1, b_2, b_3) is (without loss of generality) the output of the automaton. Projection onto initial coordinates (e.g. $(a_3, a_2, a_1) \mapsto (a_2, a_1)$ or $(a_3, a_2, a_1) \mapsto (a_1)$) is a structure-preserving mapping (homomorphism).

order (4 and 5). A group of order 4 is not simple and can be constructed from two groups of order 2. We have $PRIMES(K_1) = \{C_2, C_3, C_5\}$. For some insight into what the automatically derived coordinate structure for the Krebs cycle reveals, see Fig. 7. Again the non-trivial groups appear at the (four) deepest levels of the hierarchy with C_5, C_4, C_3, and C_2 occurring, respectively, at levels 4, 3, 2, 1 (the lowest level). For more on the interpretation of such holonomy decompositions, we refer the reader to [8], where genetic regulatory networks are considered.

4.3. *Complexity of the Krebs Cycle*

Krohn-Rhodes Theory includes a rigorous notion of complexity as the minimal number of alternations (minimal over in all possible decompositions of the structure) between levels with trivial and non-trivial group components. The complexity of finite automaton $\mathcal{A}$ is defined to be this non-negative integer number for decompositions of its associated transformation semigroup and denoted $cpx(\mathcal{A})$. This complexity measure satisfies a natural set of axioms (for an introduction and its relationship to biological mod-

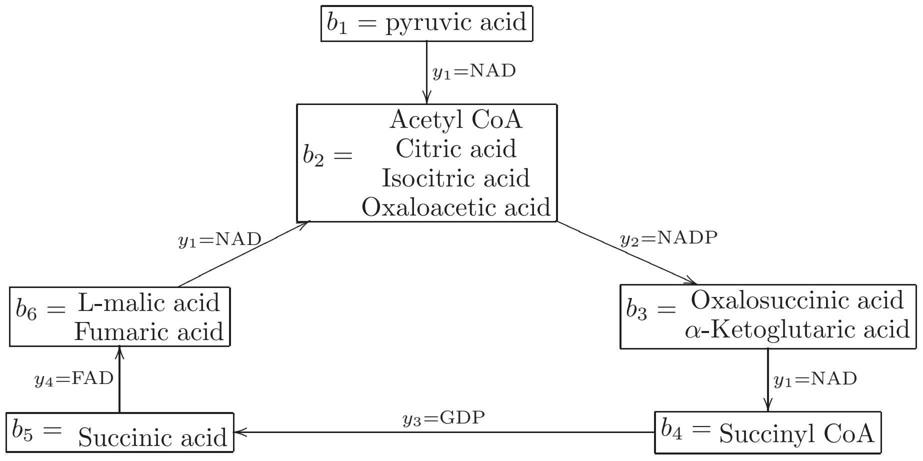
b_1 = pyruvic acid
y_1=NAD
b_2 = Acetyl CoA Citric acid Isocitric acid Oxaloacetic acid
y_1=NAD
y_2=NADP
b_6 = L-malic acid Fumaric acid
b_3 = Oxalosuccinic acid α-Ketoglutaric acid
y_4=FAD
y_1=NAD
y_3=GDP
b_5 = Succinic acid
b_4 = Succinyl CoA

Fig. 5. Simplified finite state automaton model K_2 of the Krebs cycle. This model is obtained from the model K_1 by putting into the coenzyme CoASH into the soup and then identifying the transformations due to NAD+CoASH and NAD. For instance, CoASH is needed for the reaction Acetyl CoA → {Citric acid, Isocitric acid} (i.e. $a_2 \overset{x_2}{\to} a_3$ in the original model K_1, but since CoASH has been put into the soup the reaction runs to completion and we have that all substrates of a_2 and a_3 belong to a new state b_1 of the derived model. This way we must group together several substrates as the reactions go to completion when the coenzymes are available in the soup. Moreover, putting CoASH in the soup implies that we should identify the transitions involving the same coenzymes that did and did not require CoASH. Thus, transformations x_1 =NAD+CoASH and x_6 =NAD appear as the combined transformation y_1 labelled as simply as "NAD" in the derived model.

eling and evolution, see [13]). Since in the decompositions of the two Krebs cycle models, the levels from the top until the last few levels only include trivial groups, these upper levels include no alternation with group components. The remaining, group-containing levels can be decomposed into group and non-group containing components each with a single group level. Thus these models (and holonomy decompositions in general), have complexity no more than the number of levels with non-trivial groups. It follows that

$$cpx(K_1) \leq 4 \text{ and } cpx(K_2) \leq 2,$$

and, moreover, an argument of Rhodes [3, Chapter 6, Part I] proves the dependency between the groups of K_2 is "essential" in a certain technical sense, implying that their computation could never be carried out within the same group level of *any* wreath product decomposition for K_2. This implies $cpx(K_2) \geq 2$, and thus $cpx(K_2) = 2$. The complexity of the Krebs cycle model K_1 is therefore at least 2 (since K_1 can certainly emulate K_2) but could well be as much as 4, although its exact value has not been rigorously established here.

5. Changing the BioChemical Model

We have presented two different models of the Krebs cycle, K_1 in Fig. 1 and K_2 in Fig. 5. We asserted that K_2 is a simplified version of K_1 but they are both models of the Krebs cycle. Here we make this precise.

In fact, any modeling technique should be capable of incorporating new empirical findings, refining details or simplifying some aspects of the model. Clearly, when we execute these modifications we expect some kind of gradual change in the model. For instance adding new details to one model should not give another one contradicting the original, but rather a refinement or extension of it, or, in the other direction, when simplifying the model, we expect a (possibly abstracted) substructure of it. In the automata realm this means that the division/emulation relation should be preserved through the modifications. Formally, the transformation semigroup of the larger model should have a substructure mapping in a manner that respects structure (i.e. homomorphically) onto the transformation semigroup of the smaller model. For reaction graph models we have the following operations

- **Putting a coenzyme into the soup -** in the biochemical model, this means that we assume the coenzyme is present in sufficient quantity so that all reactions in model that require that coezyme can go to completion. This corresponds algebraically to "localizing at an idempotent".
- **Unifying independent reactions -** in the biochemical reaction graph of the model, two transi-

tions with the different labels whose order of occurrence does not influence the result may be collapsed. This means that the agents acting in these transitions are no longer distinguished, i.e. they may be considered as given by the same agent, resulting in single new global transition of the system. Algebraically, this corresponds to identifying commuting transformations (generally idempotent ones in the case of reaction graphs).
- **Expanding substrates or adjoining more substrates supplied with the required enzymes**.

Mathematical proofs [3, Fact 6.9] show that the emulation relation is preserved for biochemical models for the associated transformations semigroups when applying these operations. These operations for model manipulations have been implemented in our open-source `SgpDec` software [5].

5.1. *Coenzyme into Soup*

Idempotent transformations are defined by the following property: $a \cdot t = a \cdot t \cdot t$ for all states $a \in A$, thus after applying t once, the subsequent applications of the same input would not yield any further change (at least as long as nothing else occurs in the meantime). Since reactions in our approach continue until they stabilize, their catalysts correspond to idempotents. It is very important that the cell require a different enzyme for practically every reaction it carries out. No enzyme no reaction. This specificity is not entirely absolute but for the majority of enzymes absolute specificity is the rule, therefore this technique is widely applicable when modeling biochemical reactions.

Putting a coenzyme into the soup (or, algebraically, localizing at an idempotent) is also useful computationally to reduce the size of the semigroup. Especially when we are looking for only the prime components of the automaton (such as the case of biochemical reactions), we can get some of them from the decomposition of the reduced automaton.

This operation can be formalized in the following way: Assume that for an automaton (A, X, δ) the input symbol x_0 is an idempotent, i.e. $a \cdot x_0 \cdot x_0 = a \cdot x_0$ for all states $a \in A$. Then we can construct a modified automaton (A', X', δ'), read *putting* x_0 *into the "soup" of* (A, X, δ). The new state set will consist of states that are fixed by x_0,

$$A' = \{a \in A : a \cdot x_0 = a\}.$$

Then input x_0 disappears from the new input set $X' = X - \{x_0\}$. The new state transition function is $\delta' : A' \times X' \to A'$ with

$$\delta'(a', x) = a' \cdot x_0 \cdot x \cdot x_0$$

(with $\cdot$ taken with respect to δ).

For example, in Figure 1 putting x_2 = CoASH into the soup and letting reactions run to completion entails that a_2 and a_8 are no longer observable as states since, according to the reaction graph, the reactions that yield them continue on to yield a_3 due to the presence of CoASH. Thus in the simplified model with CoASH in the soup, there are two less states, and NAD+CoASH is regarded as transforming pyruvic acid directly to citric acid, while NAD is then regarded as transforming L-malic acid to citric acid.

5.2. *Independent Reactions*

Suppose we have two different coenzymes x_1 and x_2 catalysing different reactions and their order does not matter, i.e. applying x_1 then applying x_2 has exactly the same effect on the system as applying x_2 then x_1. 'Different reactions' means that they are not connected through any substrate or product. We then say that x_1 and x_2 'commute' and we can identify them as an input. Commutativity is needed here, otherwise the identified input would yield different results depending on the order, therefore the automaton model would become nondeterministic.

Formally, for an automaton (A, X, δ), assume that $x_1, x_2 \in X$ and $a \cdot x_1 \cdot x_2 = a \cdot x_2 \cdot x_1$ for all $a \in A$. Then a modified automaton (A, X^*, δ^*), read x_1 *is identified with* x_2 in (A, X, δ) can be constructed. The state set remains the same. We replace the two commuting transformations with a new one $X^* = (X - \{x_1, x_2\}) + \{y\}$. The state transition function $\delta^* : A \times X^* \to A$ is defined with $\delta^*(a, x) = \delta(a, x)$ for $x \in X - \{x_1, x_2\}$ (just act as before for the other transformations) and

$$\delta^*(a, y) = a \cdot x_1 \cdot x_2 = a \cdot x_2 \cdot x_1$$

(replacing the commuting transformations with their unique combined transformation).

For example, after putting CoASH into the soup of the model K_1 as described above, one may identify the commuting transformations due to NAD and NAD+CoASH in the resulting model, yielding a new simplified model K_2 for the Krebs cycle (Figure 5).

5.3. *Expanding and Adjoining Substrates*

Ongoing experimental research may reveal that in a given reaction there are more substrates involved, so we would like to add them to the model together with the required enzymes/coenzymes. There are two ways to do this: either expanding an existing substrate or adding new ones.

5.3.1. *Expanding*

Let us consider the following very simple reaction network. The substrates are $\{a_1, a_2, a_3, a_4\}$ and the coenzymes are $\{x_1, x_2, x_3, x_4\}$.

$$\begin{tikzcd}
& a_1 \arrow[dr, "x_1"] & \\
a_4 \arrow[ur, "x_4"] & & a_2 \arrow[dl, "x_2"] \\
& a_3 \arrow[ul, "x_3"] &
\end{tikzcd}$$

Expanding the substrate a_1 gives the following network.

$$\begin{tikzcd}
a_1' \arrow[rr, "x"] & & a_1'' \arrow[d, "x_1"] \\
a_4 \arrow[u, "x_4"] & & a_2 \arrow[dl, "x_2"] \\
& a_3 \arrow[ul, "x_3"] &
\end{tikzcd}$$

5.3.2. *Adjoining*

Another way to add substrates is to *adjoin new substrates.* Intuitively this means adding substrates which feed into the old substrates or are fed from the old substrates, but do not form part of the basic reactions of the old substrates.

$$\begin{tikzcd}
& a_1 \arrow[dr, "x_1"] & \\
a_3 \arrow[ur, "x_3"] & & a_2 \arrow[ll, "x_2"']
\end{tikzcd}$$

After adjoining substrates and new coenzymes:

$$\begin{tikzcd}
& & & a_0 \arrow[d, "x_2"] & \\
& & & a_1 \arrow[dr, "x_1"] & \\
a_5 \arrow[r, bend left, "x_2"] & a_4 \arrow[l, "x_5"'] & a_3 \arrow[l, "x_4"'] \arrow[ur, "x_3"] & & a_2 \arrow[ll, "x_2"']
\end{tikzcd}$$

Again, this operation can be exploited for computational efficiency as well: when dealing with large networks we can remove those substrates that feed into and those that are products core reactions. Mathematically speaking, we can focus on the decomposition the strongly connected components only.

5.4. *K_2: A Derived Model*

While introducing tools for manipulating models, we have just observed how a simplified model K_2 of the Krebs cycle can be obtained from the model K_1 by the operations of putting a coenzyme into the soup and identifying commmuting inputs.

Thus Fig. 5 is a simplified reaction graph model of the Krebs cycle derived from Fig. 1. The coenzyme CoASH has been put into the soup, therefore some substrates must now be grouped together in one state, as the reactions go to completion, and inputs NAD and NAD+CoASH are identified.

As mentioned above, a theorem guarantees that the transformation semigroup associated to model K_1 has a substructure that maps in a structure-preserving way onto the transformation semigroup of model K_2. From the automated decompositions we know that $PRIMES(K_1) = \{C_2, C_3, C_5\}$ and $PRIMES(K_2) = \{C_2, C_3\}$. This again confirms the smoothness and usability of the model changing operations, and illustrates the fact that the primes in an emulated structure must be computable by the emulating one. Since the transformation semigroup of K_1 emulates the transformation semigroup of K_2, we find these groups in the primes of K_1 too.

Moreover, the second model is derived by the software from the modeler's specification of what to add and identify, and can then be immediately decomposed.

6. Discussion

We suggested an algebraic approach for modeling and analysing biochemical reactions and showed how it can be applied to automatically decompose the computational structure within real examples. After finishing the computational implementation of this modeling method, this research can now continue on a larger scale. We can start to systematically analyse biochemical reactions regarding their algebraic core structure.

The next natural questions to investigate are:
- *What predictions do the coordinate decompositions allow us to make and test for particular biochemical reactions?*

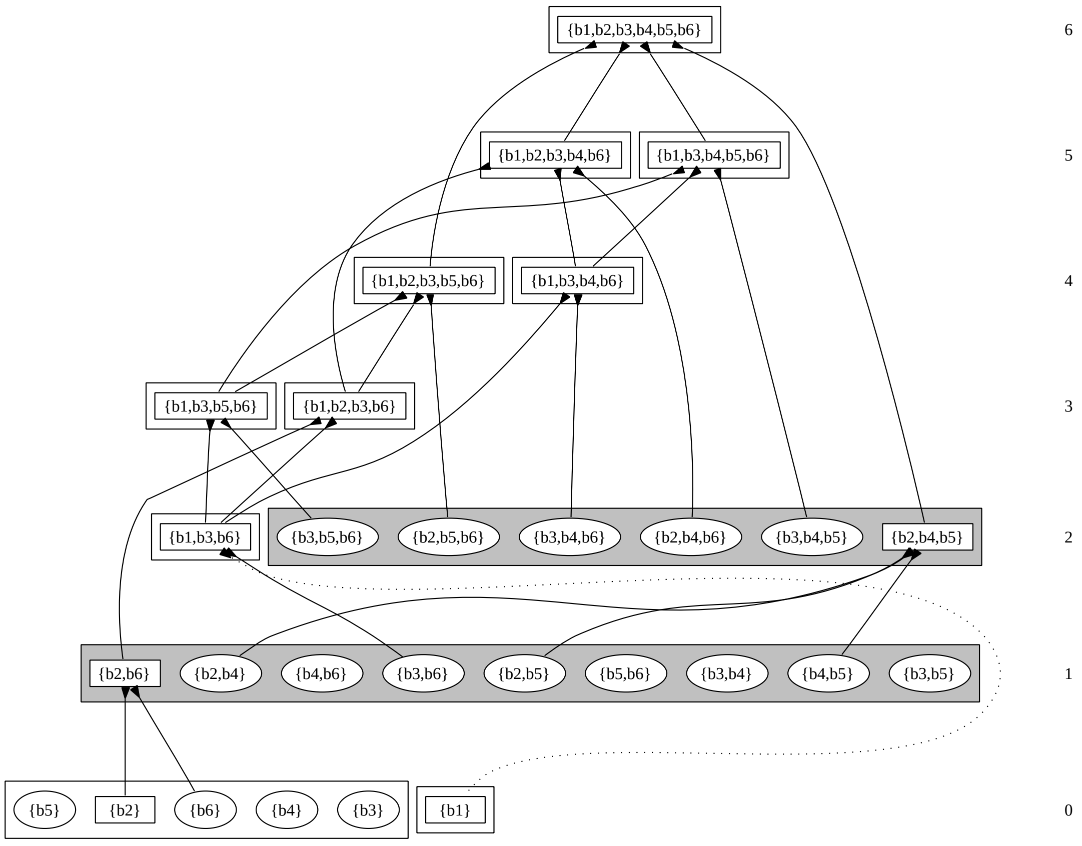

Fig. 6. The internal structure of the decomposition of the automaton model of the Krebs cycle. The numbers indicate the hierarchical levels, and the boxes on one level define the components on that coordinate position. The nodes contain subsets of the state set (i.e. the set of substrates that are observed at a given moment). The boxes indicate local reversibility, so for any two nodes within one box there is at least one sequence of coenzyme applications that takes the system from one set of substrates to the other set of substrates. Therefore the top levels denote transient states. The grey shade denotes the existence of a nontrivial permutation group component. On the first level (lowest, deepest in the hierarchy) we have a cyclic group on two points C_2, on the second level we have a cyclic group C_3 of order 3. These are the only symmetries within the system, e.g. observing the states $\{b_3, b_5, b_6\}$ and applying certain coenzymes in order (the group generators) the macroscopic system would show no change, but individual molecules would trace out the permutation group structure.

- What is the significance of the presence of permutation groups for understanding the biochemical network in questions?
- If two biochemical systems give rise to the same prime groups, how can the knowledge of one be used in understanding and manipulating the other?
- What type of simple groups can be found in the prime components of the decompositions of biochemical reactions?
And after the classes of 'biochemical groups' are found, we can ask:
-What are the special properties of these classes of simple groups?
-What makes these groups biologically embeddable?
In answering the last question, the advanced techniques of the finite group theory are expected to prove extremely helpful.

References

[1] K. Krohn, J. L. Rhodes, B. R. Tilson, The prime decomposition theorem of the algebraic theory of machines, in: M. A. Arbib (Ed.), Algebraic Theory of Machines, Languages, and Semigroups, Academic Press, 1968, Ch. 5, pp. 81–125.

[2] D. Gorenstein, R. Lyons, R. Solomon, The Classification of Finite Simple Groups, American Mathematical Society, 1994.

[3] J. L. Rhodes, Applications of Automata Theory and Algebra via the Mathematical Theory of Complexity to Biology, Physics, Psychology, Philosophy, and

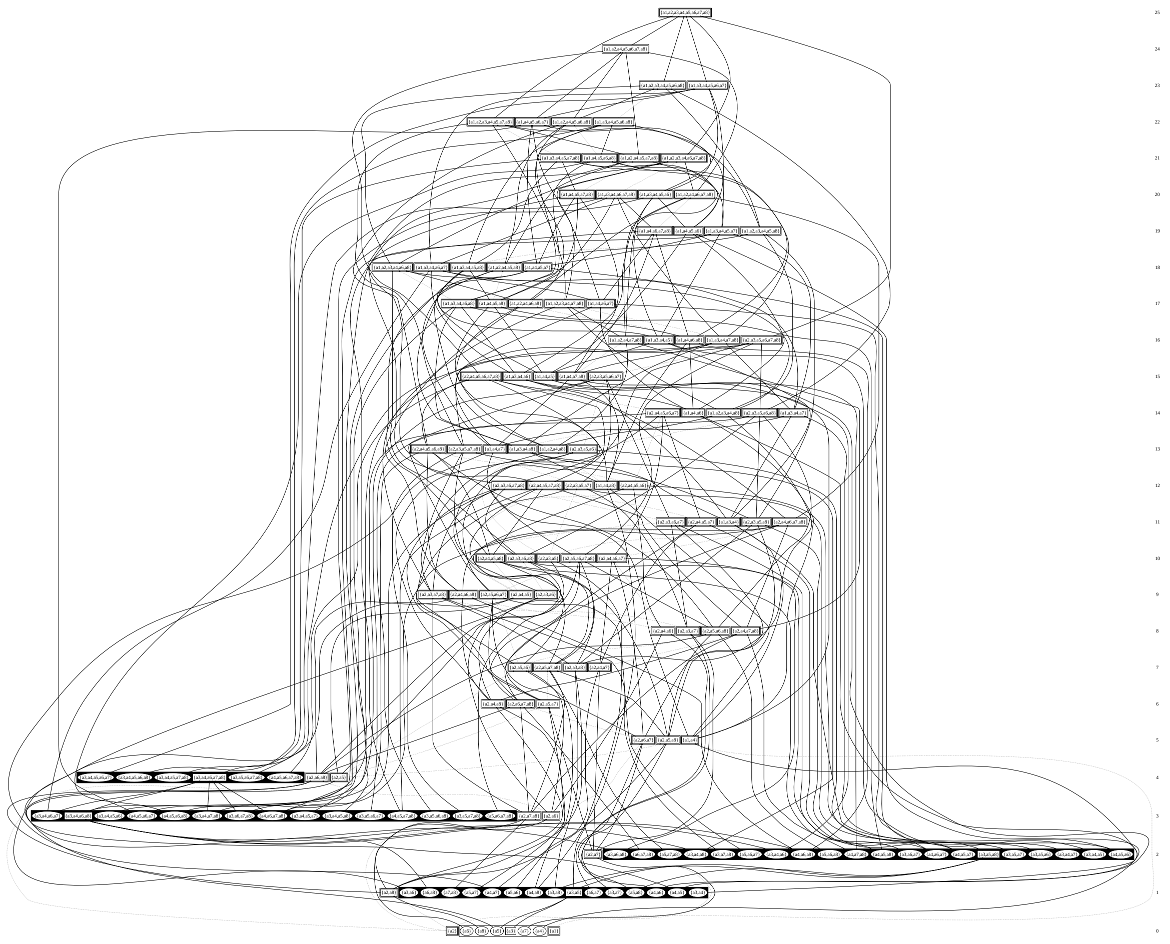

Fig. 7. The decomposition for the K_1 model of the Krebs cycle has 25 hierarchical levels and reveals four hierarchical levels involving non-trivial permutation groups (in parallel composition with other components). These non-trivial permutation group components are indicated by darker shading. The automatically generated decomposition visualized here illustrates that software tools now available to assist in understanding biochemical reaction networks can now go well beyond the realm of manual human calculations. Similarly to the K_2 model the prime components include the cyclic groups C_2 on the 1st (lowest) level and C_3 on the 2nd level as in the decomposition of K_2, but cyclic holonomy permutation groups C_4 and C_5 also appear on the 3rd and on the 4th levels, respectively. The additonal presence of C_4 and C_5 reflects the fact that K_1 is a more detailed model of the Krebs cycle. It follows that the primes of the Krebs cycle model K_1 are thus C_2, C_3, and C_5 and that its complexity is at least 2 but no more than 4 (see text). The levels of the decomposition, similar to those for K_2 in Figure 6, include transient levels (21 levels in this case), as well as macroscopic pools of stability and reversibility, corresponding to the presence of each of the non-trivial group components in the four deepest levels in the decomposition and thus reveal 'hidden' local symmetries within the Krebs cycle.

Games, World Scientific Press, in press, foreword by Morris W. Hirsch, edited by Chrystopher L. Nehaniv (Original version: University of California at Berkeley, Mathematics Library, 1971).

[4] K. Krohn, R. Langer, J. Rhodes, Algebraic principles for the analysis of a biochemical system, J. Comput. Syst. Sci. 1 (2) (1967) 119–136.

[5] A. Egri-Nagy, C. L. Nehaniv, SgpDec, jSgpDec – computational tools for semigroup decompositions, (`http://sgpdec.sf.net`)., (formerly jGrasp) (2003).

[6] A. Egri-Nagy, C. L. Nehaniv, Algebraic hierarchical decomposition of finite state automata: Comparison of implementations for Krohn-Rhodes Theory, in: Conference on Implementations and Applications of Automata CIAA 2004, Vol. 3317 of Springer Lecture Notes in Computer Science, 2004, pp. 315–316.

[7] P. Dömösi, C. L. Nehaniv, Algebraic Theory of Finite Automata Networks: An Introduction, Vol. 11, SIAM Series on Discrete Mathematics and Applications, 2005.

[8] A. Egri-Nagy, C. L. Nehaniv, Hierarchical coordinate systems for understanding complexity and its evolution with applications to genetic regulatory networks, Artificial Life 14 (3), (Special Issue on the Evolution of Complexity), in press.

[9] A. Egri-Nagy, C. L. Nehaniv, Making sense of the sensory data - coordinate systems by hierarchical decomposition, in: 10th International Conference on Knowledge-Based & Intelligent Information & Engineering Systems (KES'06), Vol. 4253 of Springer Lecture Notes in Computer Science, 2006, pp. 333–340.

[10] L. Margulis, D. Sagan, Early Life: Evolution on the PreCambrian Earth, 2nd Edition, Jones and Bartlett, 2002.

[11] J. M. Berg, J. L. Tymoczko, L. Stryer, Biochemistry, Sixth Edition : International Version, W. H. Freeman, 2006.

[12] D. J. S. Robinson, A Course in the Theory of Groups, 2nd Edition, Springer, 1995.

[13] C. L. Nehaniv, J. L. Rhodes, The evolution and understanding of hierarchical complexity in biology from an algebraic perspective, Artificial Life 6 (1) (2000) 45–67.